

**This Page is Inserted by IFW Indexing and Scanning  
Operations and is not part of the Official Record**

**BEST AVAILABLE IMAGES**

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ **BLACK BORDERS**
- ☐ **IMAGE CUT OFF AT TOP, BOTTOM OR SIDES**
- ☐ **FADED TEXT OR DRAWING**
- ☐ **BLURRED OR ILLEGIBLE TEXT OR DRAWING**
- ☐ **SKEWED/SLANTED IMAGES**
- ☐ **COLOR OR BLACK AND WHITE PHOTOGRAPHS**
- ☐ **GRAY SCALE DOCUMENTS**
- ☐ **LINES OR MARKS ON ORIGINAL DOCUMENT**
- ☐ **REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY**
- ☐ **OTHER:** \_\_\_\_\_

**IMAGES ARE BEST AVAILABLE COPY.**

**As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.**



# UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE  
United States Patent and Trademark Office  
Address: COMMISSIONER FOR PATENTS  
P.O. Box 1450  
Alexandria, Virginia 22313-1450  
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/817,670	03/26/2001	Eric Hughes	EH-2001-01	8660

7590 08/25/2004

Eric Hughes  
1577 Rose Street  
Berkeley, CA 94703

EXAMINER

NALVEN, ANDREW L

ART UNIT PAPER NUMBER

2134

DATE MAILED: 08/25/2004

Please find below and/or attached an Office communication concerning this application or proceeding.

RECEIVED

SEP 23 2004

Technology Center 2100

# Office Action Summary

Application No.

09/817,670

Applicant(s)

HUGHES, ERIC

Examiner

Andrew L. Nalven

Art Unit

2134

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

## Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If the period for reply specified above is less than thirty (30) days, a reply within the statutory minimum of thirty (30) days will be considered timely.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become **ABANDONED** (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

## Status

- 1) ☒ Responsive to communication(s) filed on 26 March 2001.
- 2a) ☐ This action is **FINAL**. 2b) ☒ This action is non-final.
- 3) ☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

## Disposition of Claims

- 4) ☒ Claim(s) 1-20 is/are pending in the application.
- 4a) Of the above claim(s) \_\_\_\_\_ is/are withdrawn from consideration.
- 5) ☐ Claim(s) \_\_\_\_\_ is/are allowed.
- 6) ☒ Claim(s) 1-20 is/are rejected.
- 7) ☐ Claim(s) \_\_\_\_\_ is/are objected to.
- 8) ☐ Claim(s) \_\_\_\_\_ are subject to restriction and/or election requirement.

## Application Papers

- 9) ☐ The specification is objected to by the Examiner.
- 10) ☒ The drawing(s) filed on 26 March 2001 is/are: a) ☒ accepted or b) ☐ objected to by the Examiner.  
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).  
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
- 11) ☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

## Priority under 35 U.S.C. § 119

- 12) ☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
- a) ☐ All b) ☐ Some \* c) ☐ None of:
- ☐ Certified copies of the priority documents have been received.
  - ☐ Certified copies of the priority documents have been received in Application No. \_\_\_\_\_.
  - ☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

\* See the attached detailed Office action for a list of the certified copies not received.

## Attachment(s)

- 1) ☒ Notice of References Cited (PTO-892)
- 2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
- 3) ☐ Information Disclosure Statement(s) (PTO-1449 or PTO/SB/08)  
Paper No(s)/Mail Date \_\_\_\_\_
- 4) ☐ Interview Summary (PTO-413)  
Paper No(s)/Mail Date. \_\_\_\_\_
- 5) ☐ Notice of Informal Patent Application (PTO-152)
- 6) ☐ Other: \_\_\_\_\_

### DETAILED ACTION

1. Claims 1-20 are pending.

#### ***Claim Rejections - 35 USC § 102***

2. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

3. Claims 1-2, 4-7, 9-14, and 16-19 are rejected under 35 U.S.C. 102(b) as being anticipated by Kocher US Patent No. 5,903,651. Kocher discloses a method for demonstrating and confirming the status of digital certificates and other data.
4. With regards to claim 1, Kocher teaches the arranging of a plurality of messages into an ordered sequence of messages (Kocher, column 6 lines 43-47 "data items...sorted in ascending order", column 11 lines 33-50), constructing a hash tree from the sequence of messages (Kocher, column 7 lines 19-26, hash tree), computing a value of a root node of a hash tree (Kocher, column 7 lines 51-54, root node), preparing a private key for a digital signature operation (Kocher, column 1 lines 46-50), and performing a cryptographic signature operation with the private key upon the value of the root node (Kocher, column 8 lines 5-13).

Art Unit: 2134

5. With regards to claims 2 and 13, Kocher teaches the tree constructed with a position dependent hash function (Kocher, column 7 lines 40-43, hash depends on level in the tree).
6. With regards to claim 4, Kocher teaches a value of the leaves of the hash tree being taken as the results of a hash function applied to the values of the sequence of messages (Kocher, column 7 lines 33-44, cryptographic hash function).
7. With regards to claims 5, 9, 11 and 18, Kocher teaches the performing of the cryptographic signature operation with padding added to the value of the root node (Kocher, column 8 lines 5-13, padding viewed as date/time stamp, number of nodes, see Figure 9 item 905).
8. With regards to claim 6, Kocher teaches all that is described above, and further teaches the extracting of the public key digital signature from a combination of the hash tree and from the results of the cryptographic signature operation (Kocher, column 9 lines 26-42).
9. With regards to claims 7 and 19, Kocher teaches the incorporating of a hash tree size into the public key digital signature where the hash tree size is the number of the plurality of messages (Kocher, column 8 lines 5-13 "number of nodes", see Figure 9 item 904).
10. With regards to claims 10, 12 and 17, Kocher teaches the parsing of the public key digital signature and the retrieving of its signature data (Kocher, column 8 lines 34-38, column 9 lines 26-36), the ascertaining that the signature data comprises a stated signature value and a stated sibling value and position sequence (Kocher, column 8

Art Unit: 2134

lines 29-32), computing a hash tree branch comprising a leaf node and a root node (Kocher, column 10 lines 1-20, column 9 lines 39-44, Figure 14), the hash tree branch being computed with the value of the individual message and with the stated sibling value and position sequence (Kocher, column 9 lines 39-44), and performing a verification operation on the stated signature value with the value of the root node and with the public key (Kocher, column 9 lines 26-52).

11. With regards to claim 14, Kocher teaches the ascertaining that the signature data comprises a hash tree size (Kocher, column 9 lines 60-61), the determining of a tree representative of the tree family and whether or not the shape of the hash tree branch is a valid branch of the tree (Kocher, column 9 line 53 – column 10 line 20).

12. With regards to claim 16, Kocher teaches the value of the leaf of the hash tree branch taken as the result of a hash function applied to the value of the individual message (Kocher, column 7 lines 33-43).

### ***Claim Rejections - 35 USC § 103***

13. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

(a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negated by the manner in which the invention was made.

Art Unit: 2134

14. Claims 3, 8, 15, and 20 are rejected under 35 U.S.C. 103(a) as being unpatentable over Kocher US Patent No. 5,903,651 in view of Ostrovosky et al US Patent No. 5,123,045.

15. With regards to claims 3, 8, 15, and 20, Kocher fails to teach the hash tree constructed using a hash function with a salt value. Ostrovosky teaches a hash tree constructed using a hash function with a salt value (Ostrovosky, column 6 lines 46-67). At the time the invention was made, it would have been obvious to a person of ordinary skill in the art to utilize Ostrovosky's method of hashing with Kocher's system because it offers the advantage of providing a random function that helps prevent attackers from corrupting memory locations (Ostrovosky, column 3 lines 28-57).

### ***Conclusion***

16. The prior art made of record and not relied upon is considered pertinent to applicant's disclosure.

17. Simon et al US Patent No. 6,065,008 discloses a system for secure font subset distribution.

18. Kaliski Jr US Patent No. 6,189,098 discloses a client/server protocol for proving authenticity.

19. Ansper et al US PG Pub 2001/0032314 discloses a method for validating a digital signature.

20. Karjoth et al US PG Pub 2001/0034839 discloses a method for secure transmission of data and applications.

21. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Andrew L Nalven whose telephone number is 703 305 8407. The examiner can normally be reached on Monday - Thursday 8-6, Alternate Fridays.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Gregory Morse can be reached on 703 308 4789. The fax phone number for the organization where this application or proceeding is assigned is 703-872-9306.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see <http://pair-direct.uspto.gov>. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

Andrew Nalven



GREGORY MORSE  
SUPERVISORY PATENT EXAMINER  
TECHNOLOGY CENTER 2100



<b>Notice of References Cited</b>	Application/Control No. 09/817,670	Applicant(s)/Patent Under Reexamination HUGHES, ERIC	
	Examiner Andrew L Naiven	Art Unit 2134	Page 1 of 1

**U.S. PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Name	Classification
	A	US-5,123,045	06-1992	Ostrovsky et al.	713/190
	B	US-5,903,651	05-1999	Kocher, Paul Carl	713/158
	C	US-6,065,008	05-2000	Simon et al.	707/10
	D	US-6,189,098	02-2001	Kaliski, Jr., Burton S.	713/168
	E	US-2001/0032314	10-2001	Ansper et al.	713/176
	F	US-2001/0034839	10-2001	Karjoth et al.	713/190
	G	US-			
	H	US-			
	I	US-			
	J	US-			
	K	US-			
	L	US-			
	M	US-			

**FOREIGN PATENT DOCUMENTS**

*		Document Number Country Code-Number-Kind Code	Date MM-YYYY	Country	Name	Classification
	N					
	O					
	P					
	Q					
	R					
	S					
	T					

**NON-PATENT DOCUMENTS**

*		Include as applicable: Author, Title Date, Publisher, Edition or Volume, Pertinent Pages)
	U	Digital Signature Flows and Multicasts, Wong et al, August 1999
	V	
	W	
	X	

\*A copy of this reference is not being furnished with this Office action. (See MPEP § 707 05(a).)  
 Dates in MM-YYYY format are publication dates. Classifications may be US or foreign.

# Digital Signatures for Flows and Multicasts

Chung Kei Wong, *Student Member, IEEE*, and Simon S. Lam, *Fellow, IEEE*

**Abstract**—We present chaining techniques for signing/verifying multiple packets using a single signing/verification operation. We then present flow signing and verification procedures based upon a tree-chaining technique. Since a single signing/verification operation is amortized over many packets, these procedures improve signing and verification rates by one to two orders of magnitude, compared to the approach of signing/verifying packets individually. Our procedures do not depend upon reliable delivery of packets. They also provide delay-bounded signing, and are thus suitable for delay-sensitive flows and multicast applications. To further improve our procedures, we propose several extensions to the Feige–Fiat–Shamir digital signature scheme to substantially speed up both the signing and verification operations, as well as to allow “adjustable and incremental” verification. The extended scheme, called eFFS, is compared to four other digital signature schemes (RSA, DSA, ElGamal, and Rabin). We compare their signing and verification times, as well as key and signature sizes. We observe that: 1) eFFS is the fastest in signing (by a large margin over any of the other four schemes) and as fast as RSA in verification (tie for a close second behind Rabin); 2) eFFS allows a tradeoff between memory and signing/verification time; and 3) eFFS allows adjustable and incremental verification by receivers.

## I. INTRODUCTION

**D**ATA confidentiality, authenticity, integrity, and nonrepudiation are basic concerns of securing data delivery over an insecure network, such as the Internet. *Confidentiality* means that only authorized receivers will get the data; *authenticity*, an authorized receiver can verify the identity of the data's source; *integrity*, an authorized receiver can verify that received data have not been modified; *nonrepudiation*, an authorized receiver can prove to a third party the identity of the data's source.<sup>1</sup>

Most investigations on securing data delivery over packet networks have focused on unicast delivery of data sent as independent packets. Exceptions include recent papers on scalable secure multicasting [1], [13], [20] and a flow-based approach to datagram security [14]. All of these papers are mainly concerned with data confidentiality.

In this paper, our concerns are data authenticity, integrity, and nonrepudiation for delay-sensitive packet flows, particularly flows to be delivered to large groups of receivers. For an

individual message (packet), these concerns can be addressed by one of many available digital signature schemes [6], [15], [17], [19]. However, these schemes are not efficient enough for signing/verifying packets individually for delay-sensitive flows, such as packet video.

In the Internet, multicast has been used successfully to provide an efficient, best-effort delivery service to large groups [2]. Consider a packet flow multicasted to a group of receivers. A consequence of best-effort delivery is that many receivers will not receive all of the packets in the multicasted flow. Furthermore, many multicast applications allow receivers to have widely varying capabilities (e.g., to receive layered video and audio transmissions) or needs (e.g., to receive different stock quotes, news, etc.). Consequently, receivers get different subsequences of packets from the same multicasted flow.

### A. Existing Techniques for Signing Flows

Conceptually, a digital signature scheme is defined by functions for key generation, signing, and verification. The signer (sender) uses the key generation function to create a pair of keys, a signing key  $k_s$ , and a verification key  $k_v$ . The signer keeps the signing key private, and makes the verification key publicly known to all verifiers (receivers).<sup>2</sup>

To sign a message  $m$  using signing key  $k_s$ , the signer calls the signing function which returns the signature of message  $m$ . The signer then sends the signed message, consisting of message  $m$  and its signature, to verifiers. Having received the signed message, a verifier calls the verification function with key  $k_v$ . If the verification function returns true, then the verifier concludes that the signer did sign the message and the message has not been altered. Moreover, the signer cannot deny having signed the message (nonrepudiation).

In practice, a message digest function, such as MD5 [18], is first applied to the message to generate a fixed-size message digest which is independent of message size. Signing a message means signing the digest of the message. (MD5 message digests are 128 bits long.)

A *flow* is a sequence of packets characterized by some attribute [16], [21]. Packets in a flow may be obtained from segmenting the bit stream of digitized video, digitized audio, or a large file. They may also be related data items, such as stock quotes, news, etc., generated by the same source.

It is easy and efficient to sign an *all-or-nothing* flow, that is, a flow whose entire content is needed before any part of it can be used, e.g., a long file. In this case, the signer simply generates a message digest of the entire flow (file) and signs the message digest.

<sup>2</sup>The signing and verification keys are also referred to as private and public keys, respectively.

Manuscript received January 27, 1999; revised June 14, 1999; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor J. Crowcroft. This work was supported in part by Texas Advanced Research Program under Grant 003658-063 and by the NSA INFOSEC University Research Program under Grant MDA904-98-C-A901. An earlier version of this paper appears in *Proceedings IEEE ICNP '98*.

The authors are with the Department of Computer Sciences, the University of Texas at Austin, Austin, TX 78712-1188 USA (e-mail: ckwong@computer.org; lam@cs.utexas.edu).

Publisher Item Identifier S 1063-6692(99)07274-X.

<sup>1</sup>In the balance of this paper, we use “receiver” to mean “authorized receiver” unless otherwise stated.

Most applications, however, create flows that are not all-or-nothing. That is, a receiver needs to verify individual packets (or, more generally, application data units) and use them before the entire flow is received. For these flows, a straightforward solution is to sign each packet individually and each packet is verified individually by receivers. This solution is called the *sign-each* approach.

The sign-each approach is computationally expensive. The signing rate and verification rate are at most  $1/(T_d(l) + T_{\text{sign}})$  and  $1/(T_d(l) + T_{\text{verify}})$  packets/s, respectively, where  $T_d(l)$  is the time to compute the message digest of an  $l$ -byte packet,  $T_{\text{sign}}$  is signing time, and  $T_{\text{verify}}$  is verification time for the message digest of a packet. The signing and verification rates,<sup>3</sup> in packets/s, of two widely used digital signature schemes, RSA [19]<sup>4</sup> and DSA [15], with 512-bit modulus and using 100% processor time of a Pentium II 300-MHz machine are shown below:

packet size (bytes)	Signing rate		Verification rate	
	RSA	DSA	RSA	DSA
512	78.8	176	2180	128
1024	78.7	175	1960	127
2048	78.0	172	1620	126

If a slower machine is used, or only a fraction of processor time is available for signing/verification (e.g., a receiver machine has only 20% processor time for verification because the other 80% is needed for receiving and processing packets), then the rates should be decreased proportionally.

The signing rate is not important for a *nonreal-time generated* flow, i.e., a flow whose entire content is known in advance (such as stored video). This is because packets in the flow can be signed in advance. For a real-time generated flow, however, the signing rate must be higher than the packet-generation rate of the flow. Furthermore, for delay-sensitive flows, real-time generated or not, the verification rate is important. From the above table, we see that the signing and verification rates of the sign-each approach, using either RSA or DSA, are probably inadequate for many applications.

Two techniques were previously proposed for signing digital streams [7] which, at first glance, may be used for signing packet flows. To describe the technique in [7] for signing a nonreal-time generated flow, consider a sequence of  $m$  packets. The sender first computes message digest  $D_m$  of packet  $m$  (the last packet) and concatenates packet  $m-1$  and  $D_m$  to form augmented packet  $m-1$ . Then, for  $i = 1, \dots, m-2$ , the sender computes message digest  $D_{m-i}$  of augmented packet  $m-i$ , and concatenates packet  $m-i-1$  and  $D_{m-i}$  to form augmented packet  $m-i-1$ . Message digest  $D_1$  of augmented packet 1 is computed and signed. In this manner, only one expensive signing/verification operation is needed for the sequence of  $m$  packets. However, a necessary condition for using the above technique is the following *get-all-before* requirement: To verify packet  $i$  in the sequence, a

receiver must have received every packet from the beginning of the sequence.

For a real-time generated flow, a similar technique is suggested in [7] with the same get-all-before requirement. For a sequence of  $m$  packets, only one expensive signing/verification operation is needed, plus one inexpensive *one-time signature* signing/verification for each packet in the sequence. However, since one-time signatures and keys are very large, this technique has a large communication overhead (around 1000 bytes/packet) [9], [10].

The get-all-before requirement of both techniques in [7] is too strong for practical Internet applications. Reliable packet delivery is not used by many applications for flows and multicasts. For example, reliable delivery is generally not used for video and audio flows due to the extra delays associated with retransmissions; either losses are tolerated or forward error correction techniques are used instead.

For large-scale multicast applications, reliable delivery of multicast packets is a difficult problem [5]. Moreover, even if reliable multicasting is available, receivers with different needs/capabilities may choose to get different subsequences of packets in a multicasted flow. In short, the get-all-before requirement is not satisfied.

## B. Characteristics and Requirements

We have observed various characteristics in the delivery of flows and multicasts by an unreliable packet network, such as the Internet. They are summarized below.

- Each packet in a flow may be used as soon as it is received.
- A receiver may get only a subsequence of the packets in a flow. Different receivers may get different subsequences.
- Delay sensitive flows require fast processing at receivers. Real-time generated flows require fast processing at senders as well.
- For a multicasted flow, many receivers are limited in resources (processing capacity, memory, communication bandwidth, etc.) compared to the sender, which is typically a dedicated server machine. In some environments, both senders and receivers may be limited in resources, e.g., mobile computers using wireless communications.
- Receivers may have widely different capabilities/resources. For example, receivers may be personal digital assistants, notebook computers, or desktop machines. Moreover, the resources available to a receiver for verifying signatures may vary over time.

Given the above characteristics, we design procedures for signing and verifying flows in Section II, as well as a digital signature scheme in Section III to meet the following requirements.

- The signing procedure is efficient and, for real-time generated flows, delay bounded.
- The verification procedure is efficient (since many receivers have limited resources).
- Packets in a flow are *individually verifiable*.
- Packet signatures are small (i.e., small communication overhead).

<sup>3</sup>The signing and verification rates are rates for signing and verifying 128-bit MD5 message digests of packets.

<sup>4</sup>In this paper, we use  $e = 3$  in RSA to obtain its fastest verification time without affecting its signing time.

- *Adjustable and incremental verification:* The verification operation is adjustable to the amount of resources a receiver has. It allows a receiver/verifier to verify a message at a lower security level using less resources, and later increase the security level by using more resources (e.g., if the message is important).

### C. Contributions of this Paper

In Section II, we first describe and compare two chaining techniques (star and tree) for signing/verifying multiple packets using a single signing/verification operation (without the get-all-before requirement in [7]). We then present flow signing and verification procedures based upon the tree-chaining technique. Since a single signing/verification operation is amortized over many packets, these procedures improve signing and verification rates by one to two orders of magnitude compared to the sign-each approach. The signing procedure also provides delay-bounded signing. Thus the procedures can be used for delay-sensitive flows.

Since signed packets in our procedures are individually verifiable, the procedures can be used to reduce the workload of any machine that sends out a large number of signed packets to one or more destinations. There is no requirement that these packets belong to flows. However, for packets that belong to a flow, the workload of the flow's receiver(s) is also reduced.

In Section III, we turn our attention to improving the signing and verification operations in the procedures. Specifically, we present several extensions to the Feige-Fiat-Shamir digital signature scheme to speed up both signing and verification as well as to allow adjustable and incremental verification. In Section IV, the extended Feige-Fiat-Shamir (eFFS) scheme is compared to four well-known signature schemes [6], [15], [17], [19]. We compare their signing and verification times, as well as key and signature sizes. We observe that: 1) eFFS is the fastest in signing (by a large margin over any of the other four schemes) and as fast as RSA in verification (tie for a close second behind Rabin); 2) eFFS allows a tradeoff between memory and signing/verification time; and 3) eFFS allows adjustable and incremental verification by receivers.

## II. HOW TO SIGN A FLOW

To digitally sign/verify delay-sensitive flows, the sign-each approach is computationally too expensive for many applications, particularly those applications that generate packet flows in real time.

As an alternative to the sign-each approach, we present two chaining techniques (star and tree) for providing authenticity to a group of packets, called a *block*, using a single signing operation. The basic idea is to compute a block digest which is signed. In order to make packets *individually verifiable*, each packet needs to carry its own authentication information consisting of the signed block digest (*block signature*) together with some chaining information as proof that the packet is in the block.

### A. Star Chaining

Consider  $m$  packets that constitute a block. In star chaining, the block digest is simply the message digest of the  $m$  packet

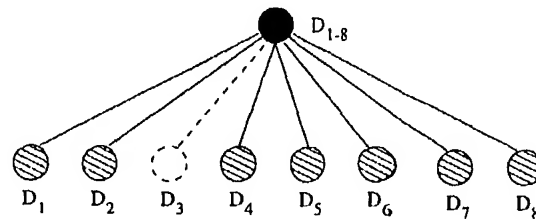


Fig. 1. Star-chaining technique.

digests (listed sequentially). Let  $h(\cdot)$  denote the message digest function being used (e.g., MD5). Consider, for example, a block of eight packets with packet digests  $D_1, \dots, D_8$ . The block digest is  $D_{1-8} = h(D_1, \dots, D_8)$ , and the block signature  $sign(D_{1-8})$  is the block digest signed with some digital signature scheme (such as RSA, DSA, or eFFS).

The relationship between the packet digests and the block digest can be represented by a one-level rooted tree, called an *authentication star*. Fig. 1 illustrates an authentication star for eight packets, with packet digests at leaf nodes, and the block digest at the root.

For packets to be individually verifiable, each packet needs its own authentication information. Such authentication information, called *packet signature*, consists of the block signature, the packet position in the block, and the digests of all other packets in the block. (We use the term *chaining overhead* to refer to all information in a packet signature except the block signature.)

Suppose the third packet in the above example is received. Its authenticity can be individually verified as follows. The verifier computes the digest  $D'_3$  of the packet received, and then the block digest  $D'_{1-8} = h(D_1, D_2, D'_3, D_4, \dots, D_8)$ , where  $D_1, D_2, D_4, \dots, D_8$  are carried in the packet signature. The verifier then calls the verification operation to verify  $D'_{1-8}$ , i.e., to determine whether  $D'_{1-8}$  is equal to block digest  $D_{1-8}$  in block signature  $sign(D_{1-8})$ . The packet is verified if the verification operation returns true, i.e.,  $D'_{1-8} = D_{1-8}$ .

Suppose the third packet is the first in the block to arrive and its authenticity has been verified. Afterwards, the verifier knows every node in the authentication star, i.e., all nodes in the authentication star are verified and can be cached. With caching, when another packet in the block arrives later, say the sixth packet, the verifier only needs to compute the digest  $D'_6$  of the packet received and compare it to the verified node  $D_6$  in the authentication star. If they are equal, the packet is verified.

### B. Tree Chaining

Tree chaining subsumes star chaining as a special case. With tree chaining, the block digest is computed as the root node of an *authentication tree*.<sup>5</sup> Consider, for example, a block of eight packets with packet digests  $D_1, \dots, D_8$ . The packet digests are the leaf nodes of a degree-two (binary) authentication tree, with other nodes of the tree computed as message digests of

<sup>5</sup>Tree chaining was first presented in [11]. Any rooted tree can be used as an authentication tree with packet digests at leaf nodes and the block digest at the root. In particular, there is no need to use a balanced tree.

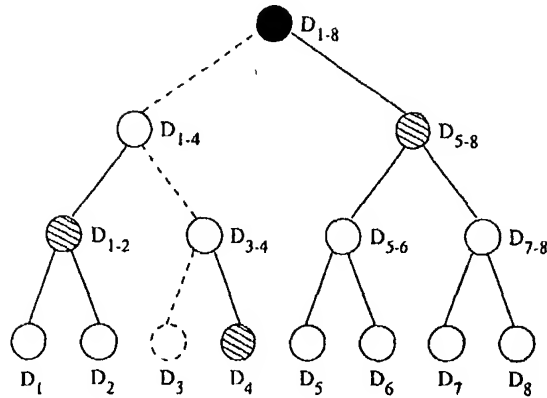


Fig. 2. Tree-chaining technique.

their children, as shown in Fig. 2. For example, the parent of the leaves  $D_1$  and  $D_2$  is  $D_{1-2} = h(D_1, D_2)$ , where  $h(\cdot)$  is the message digest function being used. The root is the block digest, with the block signature being the signed block digest.

For a packet to be individually verifiable, each packet needs to carry its own authentication information (packet signature). In tree chaining, a packet signature consists of the block signature, the packet position in the block, and the siblings of each node in the packet's path to the root. (Again, we use the term *chaining overhead* to denote all information in a packet signature except the block signature.)

To verify a packet individually, a verifier needs to verify its path to the root. Consider, for example, the dashed path in Fig. 2 for the third packet. Each node in the path needs to be verified. A verifier computes the digest  $D'_3$  of the received packet, and then each of its ancestors in the tree. That is,  $D'_{3-4} = h(D'_3, D_4)$ ,  $D'_{1-4} = h(D_{1-2}, D'_{3-4})$ , and  $D'_{1-8} = h(D'_{1-4}, D_{5-8})$ , where  $D_4$ ,  $D_{1-2}$  and  $D_{5-8}$  are carried in the packet signature. The verifier then calls the verification operation to determine whether  $D'_{1-8}$  is equal to block digest  $D_{1-8}$  in block signature  $sign(D_{1-8})$ . The packet is verified if the verification operation returns true, i.e.,  $D'_{1-8} = D_{1-8}$ .

Suppose the third packet is the first in the block to arrive. After verifying it, the verifier knows the following nodes<sup>6</sup> in the authentication tree:  $D_3$ ,  $D_4$ ,  $D_{1-2}$ ,  $D_{3-4}$ ,  $D_{1-4}$ ,  $D_{5-8}$  and the block digest  $D_{1-8}$ . These are verified nodes which can be cached. By caching verified nodes, the verifier only needs to compute each node in the authentication tree at most once.

For example, after verifying the third packet, to verify the sixth packet which arrives later, the verifier computes the digest of the packet received  $D'_6$ , its parent  $D'_{5-6} = h(D_5, D'_6)$ , and its grandparent  $D'_{3-8} = h(D'_{5-6}, D_{7-8})$ . If  $D'_{3-8}$  is equal to the cached node  $D_{3-8}$ , the sixth packet is verified.

### C. Comparison of Chaining Techniques

We performed experiments on a Pentium II 300-MHz machine running Linux, and compared star and tree chaining. We used MD5 as the message digest function [18] for generating

<sup>6</sup>Some are carried in the packet signature and the others have been computed.

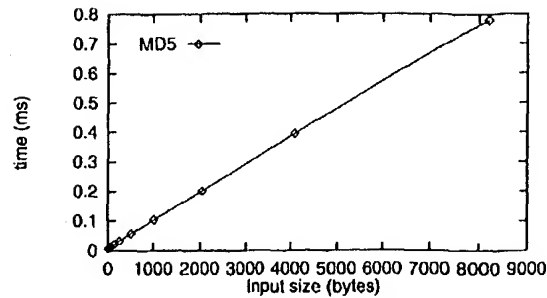


Fig. 3. MD5 computation time (milliseconds).

128-bit message digests. Fig. 3 shows the MD5 computation time versus input size. We observe that the MD5 computation time can be regarded as a linear function in input size (for a large input, i.e., 1024 bytes or more).

For each chaining technique, an authentication tree is first built for a block of packets,<sup>7</sup> i.e., each node is computed as the message digest of its children. The time to build an authentication tree (excluding time to compute packet digests for leaf nodes) is called the *tree build time*. The block signature is then obtained by signing the block digest at the root. After that, the packet signature of each packet is built from the authentication tree and the block signature. The time to build a packet signature is called *packet signature build time*. The *chaining time* for a block at a signer is the sum of tree build time and packet signature build time for all packets in the block.<sup>8</sup> Fig. 4(a) shows the chaining time for a block of packets at a signer.

Consider the total signing time for all packets in a block, which is the block's chaining time plus the signing time of the block digest. The block digest signing time is 12.7 ms using 512-bit RSA and 5.6 ms using 512-bit DSA. For a block of 16 packets, from Fig. 4(a), the chaining time is 0.21 ms for a degree-two authentication tree. The total signing time is  $0.21 + 12.7 = 12.9$  ms using 512-bit RSA. Thus, the average signing time for one packet is  $12.9/16 = 0.81$  ms, which is less than 1/15 of the block digest signing time using 512-bit RSA.

To verify packets in a block, an authentication tree is built from packet signatures as packets arrive. The *chaining time* for a block at a verifier is the sum of tree build time and time to verify chaining information in the packet signature of every packet in the block.<sup>9</sup> Fig. 4(b) shows the chaining time for a block of packets at a verifier with caching of verified nodes.

Consider the total verification time for all packets in a block, which is the block's chaining time plus the verification time of the block signature. The signature verification time is 0.40 ms using 512-bit RSA and 7.6 ms using 512-bit DSA. For a block of 16 packets, from Fig. 4(b), the chaining time is 0.24 ms for a degree-two authentication tree. The total verification time is

<sup>7</sup>We will use "tree" instead of "tree-star" since star chaining is a special case of tree chaining.

<sup>8</sup>Note that chaining time does not include time to compute packet digests for leaf nodes and time to sign the block digest.

<sup>9</sup>Note that chaining time does not include time to compute packet digests for leaf nodes and time to verify the block signature.

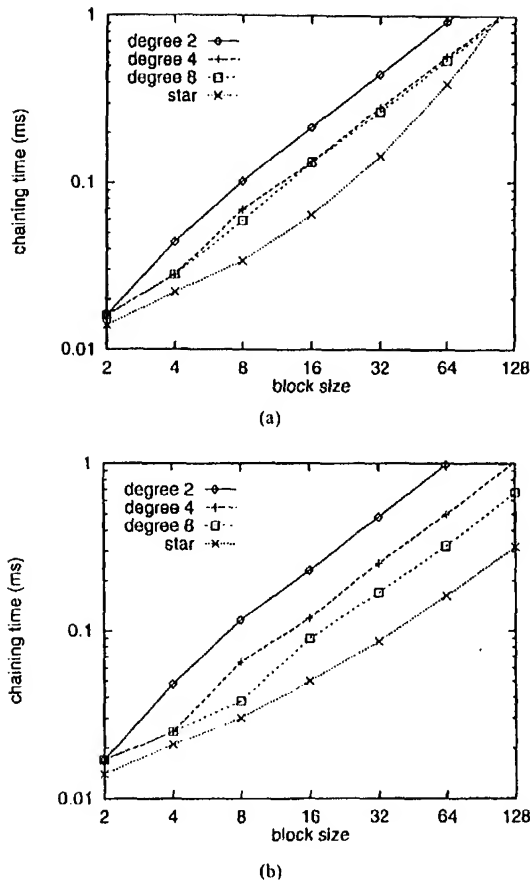


Fig. 4. Chaining time (milliseconds) for a block (a) at a signer and (b) at a verifier (with caching of verified nodes).

$0.24 + 0.40 = 0.64$  ms using 512-bit RSA. Thus the average verification time for one packet is  $0.64/16 = 0.04$  ms, which is  $1/10$  of the signature verification time using 512-bit RSA.

From Fig. 4(a), note that for any block size smaller than or equal to 64 packets, star chaining takes less time at a signer than tree chaining (degrees two to eight). However, for a larger block size, star chaining takes more time at a signer than tree chaining, because the chaining time for a star is  $O(m^2)$  and the chaining time for a tree is  $O(m \log(m))$ , where  $m$  denotes block size.

As shown in Fig. 4(b), star chaining takes less time at a verifier than tree chaining for all block sizes.

For each chaining technique, a packet signature has two parts, the block signature and the chaining signature. In general, if a tree is not balanced and full, the chaining overhead sizes of different packets are different. Fig. 5 shows the average chaining overhead size per packet. The size of the block signature is not included in Fig. 5 since it depends on which signature scheme is used (e.g., the block signature is 64 bytes for 512-bit RSA, and 40 bytes for 512-bit DSA).

From Fig. 5, note that the chaining overhead of star chaining is much greater than tree chaining for block sizes larger than eight. If a small communication overhead is important, packet signature sizes should be reduced. We recommend the use of

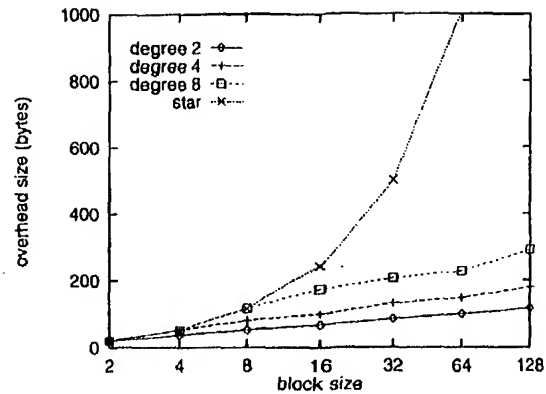


Fig. 5. Average chaining overhead size (bytes) per packet.

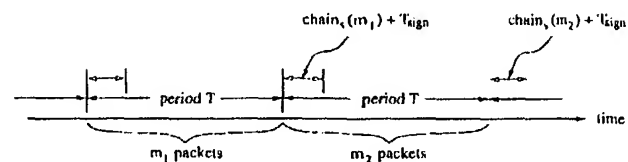


Fig. 6. Signing a real-time generated flow.

degree-two tree chaining which requires the smallest chaining overhead. (From Fig. 4, a degree-two tree has a slightly higher chaining time than the alternatives, but the difference is insignificant because chaining time is much smaller than signing/verification time of the block digest. See Figs. 7 and 8 in Section II-D.)

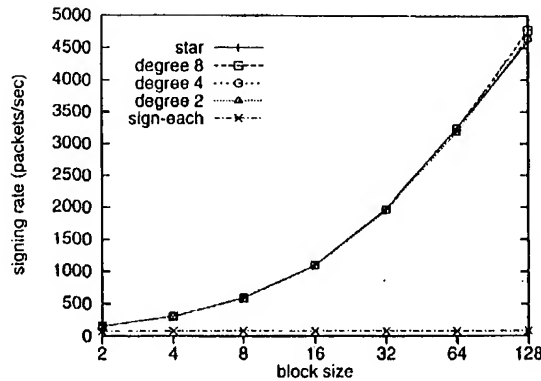
#### D. Flow Signing and Verification Procedures

A flow is signed by partitioning it into blocks of packets, with each block signed using tree chaining. For a nonreal-time generated flow, blocks are of the same size  $m$ , chosen to be a power of the authentication tree degree  $d$ . For a real-time generated flow, the packet generation rate is time-varying for many applications, such as compressed video and voice-activated audio. For these applications, partitioning the flow into fixed-size blocks may lead to an unpredictable (perhaps unbounded) signing delay. Instead, the flow is partitioned by fixed time periods, and packets generated in the same time period are grouped into a block (see Fig. 6).

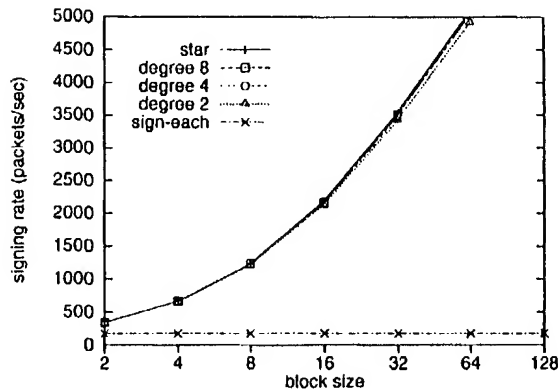
For both real-time and nonreal-time generated flows, the flow verification procedure is the same. For the first received packet in a block, i.e., the block signature carried in the packet signature is new to a verifier, the verifier computes the packet digest, and every ancestor of the packet digest.<sup>10</sup> For the computed block digest (the root of authentication tree), the verifier calls the verification operation to verify that it is equal to the block digest in the block signature. If so verified, then all computed nodes and their children are verified and cached.

For a packet that is not the first received packet in a block, the verifier computes the packet digest. If the packet digest has been cached and the cached value is equal to the

<sup>10</sup> An ancestor node is computed as the message digest of its children which are either computed or carried in the packet signature.



(a)



(b)

Fig. 7. Flow signing rate (packets/s) for 1024-byte packets. (a) Using 512-bit RSA. (b) Using 512-bit DSA.

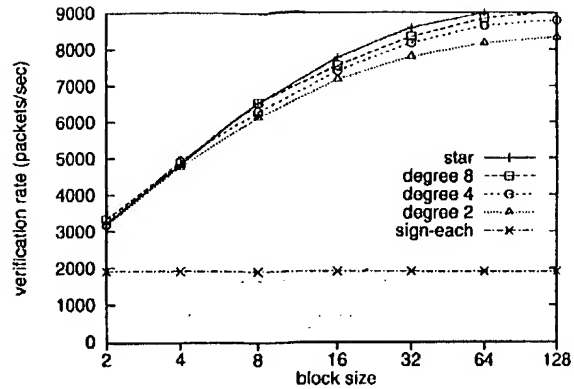
computed packet digest, then the packet is verified. Otherwise, the verifier computes every noncached ancestor of the packet digest. For the highest noncached node computed, the verifier then computes its parent. If the computed parent and its cached value are equal, the packet is verified and all computed nodes and their children are verified and cached.

We implemented the flow signing and verification procedures (see Appendix) and performed experiments on a Pentium II 300-MHz machine running Linux. We used MD5 as the message digest function, and experimented with both 512-bit RSA and 512-bit DSA as the signature scheme for block signatures.

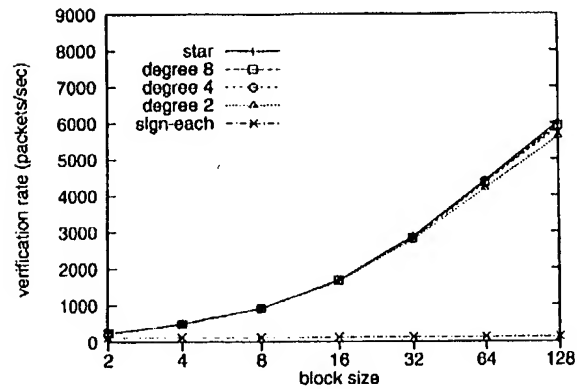
Figs. 7 and 8 show, respectively, the flow signing and verification rates for 1024-byte packets.<sup>11</sup> Note that tree and star chaining are one to two orders of magnitude more efficient than the sign-each approach. The flow signing and verification rates increase with block size. However, the rates vary only slightly with the chaining technique used and with the tree degree in tree chaining. Since degree-two tree chaining has the lowest chaining overhead (packet signature size), we recommend the use of degree-two tree chaining.

Figs. 9 and 10 show, respectively, the flow signing and verification rates for packets of size 512, 1024, or 2048 bytes. We used degree-two tree chaining. From the figures, observe

<sup>11</sup> Verification rates were computed assuming no packet loss.



(a)



(b)

Fig. 8. Flow verification rate (packets/s) for 1024-byte packets. (a) Using 512-bit RSA. (b) Using 512-bit DSA.

that the flow signing and verification rates decrease as the packet size increases. It is because more time is needed to compute the message digest of a larger packet. The decrease is more pronounced when the block size used is large, since more time is used to compute packet digests for a large block than a small block. Observe also that the flow signing and verification rates increase with block size and the increase is greater for a smaller packet size.

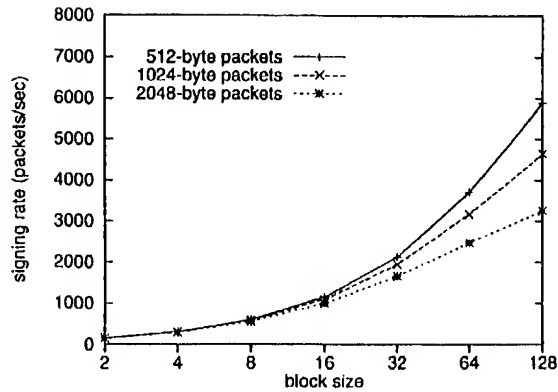
#### E. Bounded Delay Signing

Consider Fig. 6. Assume that, in period  $T$ , at most  $m$  packets are generated and their packet digests computed. The time for signing a block of  $m$  packets is  $\text{chain}_s(m) + T_{\text{sign}}$  where  $\text{chain}_s(m)$  is the chaining time for a block of  $m$  packets at a signer, and  $T_{\text{sign}}$  is the block digest signing time. Therefore, the delay of any packet within the block is at most  $D_s = T + \text{chain}_s(m) + T_{\text{sign}}$ .

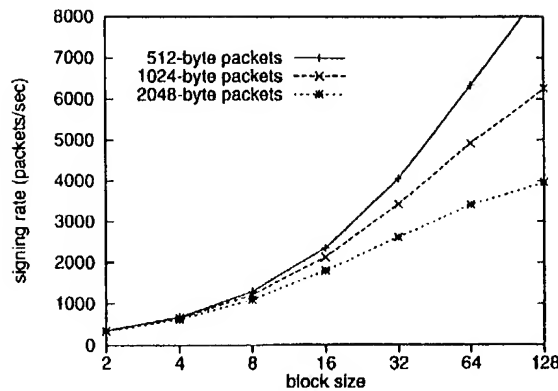
Table I shows the delay upper bound  $D_s$  for period  $T = 50$  ms. Note that the upper bound is fairly insensitive to block size since the block's chaining time is much smaller than the block digest signing time.

For a given application with a specified upper bound,  $D_s$ , for signing a real-time generated flow at a known packet rate, we can work backward and derive an appropriate value for the parameter  $T$  needed for the signing procedure of a real-





(a)



(b)

Fig. 9. Flow signing rate (packets/s) for degree-two tree chaining. (a) Using 512-bit RSA. (b) Using 512-bit DSA.

TABLE 1  
SIGNING DELAY BOUND (MILLISECONDS) FOR PERIOD  
 $T = 50$  ms USING 512-BIT RSA

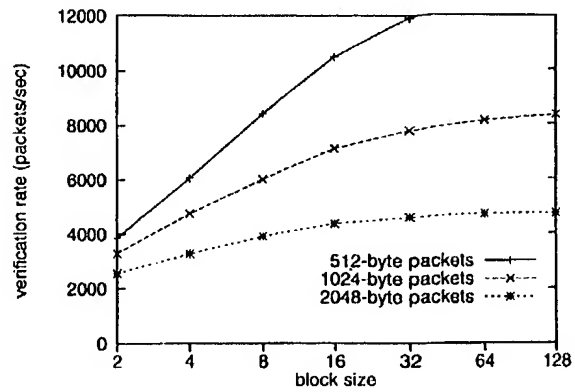
	number of packets generated in period $T$						
	2	4	8	16	32	64	128
tree deg 2	62.9	62.9	62.9	63.1	63.3	63.8	64.9
tree deg 4	62.8	62.9	62.9	63.0	63.2	63.5	64.2
tree deg 8	62.8	62.9	62.9	63.0	63.2	63.5	64.2

time generated flow. Observe, from Fig. 6, that  $T$  must be larger than  $T_{\text{sign}} + \text{chain}_s(m)$ , and  $D_s$  must be larger than  $2(T_{\text{sign}} + \text{chain}_s(m))$ .

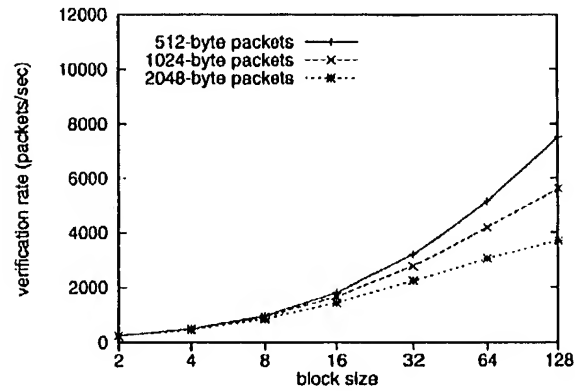
#### F. Selecting a Digital Signature Scheme

For nonreal-time generated flows, signing efficiency is not critical. Thus a signature scheme with an efficient verification operation, such as RSA, can be used in the flow signing and verification procedures. For real-time generated flows, however, it is critical that both signing and verification are highly efficient. Furthermore, in choosing a digital signature scheme, we must also consider machine capabilities (sender and receiver), as well as the fraction of processor time available for signing and verification.

Using 100% processor time of a Pentium II 300-MHz machine, the flow signing and verification rates for 1024-byte



(a)



(b)

Fig. 10. Flow verification rate (packets/s) for degree-two tree chaining. (a) Using 512-bit RSA. (b) Using 512-bit DSA.

packets, degree-two tree chaining, and block size 16 are shown below:

	signing rate	verification rate
512-bit RSA	1090 packets/sec	7030 packets/sec
512-bit DSA	2140 packets/sec	1660 packets/sec

Note that using DSA, the flow verification rate is smaller than the flow signing rate. This is undesirable because receivers/verifiers are generally less powerful than the signer/sender, e.g., the receivers may be personal digital assistants or low-end notebook computers. Using RSA, the flow signing rate may not be high enough for some applications. Although we can increase the flow signing and verification rates by using a longer period or a larger block size, neither option is desirable. A larger block size increases the chaining overhead (packet signature size). A longer period increases the delay for signing real-time generated flows.

To obtain a signature scheme better than RSA and DSA for signing/verifying flows, we propose several extensions to the Feige-Fiat-Shamir (FFS) signature scheme. The extended scheme, called eFFS, is presented in the next section. The eFFS scheme has a very efficient signing operation (much more efficient than those of RSA and DSA) and a verification operation as efficient as that of RSA. A performance compari-



son of eFFS with five other signature schemes (including FFS, RSA and DSA) is given in Section IV.

### III. THE eFFS SIGNATURE SCHEME

In Section III-A, we first describe the basic FFS signature scheme [3], [4]. The eFFS signature scheme is derived from FFS with two kinds of extensions. Three extensions to speed up the signing and verification operations of FFS are presented in Section III-B. An extension to provide *adjustable* and *incremental* signature verification is presented in Section III-C.

#### A. Basic FFS Scheme

In the basic FFS signature scheme with parameter  $(k, t)$  [3], [4], each signer chooses two large primes  $p$  and  $q$ , and computes modulus  $n = pq$ . Then, the signer chooses  $k$  integers  $v_1, \dots, v_k$  (or  $k$  integers  $s_1, \dots, s_k$ ), and computes  $s_1, \dots, s_k$  (or  $v_1, \dots, v_k$ ) by  $s_i^2 = v_i^{-1} \bmod n$ . The signing key is  $\{s_1, \dots, s_k, n\}$  and the verification key is  $\{v_1, \dots, v_k, n\}$ .

To sign message  $m$ , the signer does the following steps: 1) choose  $t$  random integers,  $r_1, \dots, r_t$ , between 1 and  $n$ , and compute  $x_i = r_i^2 \bmod n$  for  $i = 1, \dots, t$ ; 2) calculate the message digest  $h(m, x_1, \dots, x_t)$  where the message digest function  $h(\cdot)$  is public knowledge and the message digest is at least  $k \times t$  bits long; let  $\{b_{ij}\}$  be the first  $k \times t$  bits of the message digest where  $i = 1, \dots, t$ , and  $j = 1, \dots, k$ ; 3) compute  $y_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod n$  for  $i = 1, \dots, t$ . The signature of message  $m$  consists of  $\{y_i\}$  for  $i = 1, \dots, t$  and  $\{b_{ij}\}$  for  $i = 1, \dots, t$  and  $j = 1, \dots, k$ .

To verify the signature of message  $m$ , a verifier computes  $z_i = y_i^2 \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = 1, \dots, t$ . The signature is valid if and only if the first  $k \times t$  bits of  $h(m, z_1, \dots, z_t)$  are equal to the  $\{b_{ij}\}$  received. (It can be shown that  $z_i$  computed by the verifier is equal to  $x_i$  at the signer.)

The security level of FFS( $k, t$ ) depends on the following: 1) the size of modulus  $n$  (i.e., the size of the primes  $p$  and  $q$ ) and 2) the value of product  $kt$ . A system with a larger modulus is more secure, and a system with a larger  $kt$  product is more secure. If two systems have the same modulus and same  $kt$  product (but different  $k$  and  $t$  values), then their security levels are about the same.

Assuming  $|v_i| = |n|$  and  $|s_i| = |n|$ , where  $|x|$  denotes the size of  $x$  in bits, the signing/verification key size is  $(k+1) \times |n|$  bits, and the signature size is  $t \times |n| + k \times t$  bits. The signing/verification key size only depends on  $k$ , but the signature size is proportional to  $t$ . Thus, for a fixed  $kt$  product, we can reduce the signature size by using a smaller  $t$  (and a larger  $k$ ). For  $t = 1$ , the signature size is minimized, but the signing/verification key size is maximized. Table II shows the signing/verification key size and signature size of FFS with 512-bit modulus.

#### B. Extensions to Speed up FFS

1) *Small Verification Key (small v-key)*: In FFS, the sizes of signing key components  $\{s_i\}$  affect the signing time, and the sizes of verification key components  $\{v_i\}$  affect the verification time. An improvement suggested in [12] is to

TABLE II  
FFS SIGNING/VERIFICATION KEY AND SIGNATURE  
SIZES (BYTES) WITH 512-BIT MODULUS

	$t = 1$		$t = 2$		$t = 4$	
	key	sig	key	sig	key	sig
$kt = 64$	4160	72	2112	136	1088	264
$kt = 128$	8256	80	4160	144	2112	272

use the first  $k$  prime numbers as verification key components  $\{v_i\}$ . However, since not every prime number  $p$  satisfies the condition that there exists an integer  $s$  such that  $s^2 = p^{-1} \bmod n$ , we propose to use the first  $k$  prime numbers that satisfy this condition as verification key components.<sup>12</sup> This extension reduces both the verification time and the verification key size.

2) *Chinese Remainder Theorem (crt)*: The signing operation in FFS involves the computing of  $y_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod n$ . For  $n = pq$ , from the Chinese Remainder Theorem, a signer can compute  $y_i$  from  $a_i$  and  $b_i$  using the following formula:  $y_i = ((a_i - b_i) \times q \times q_p^{-1} + b_i) \bmod n$  where  $q_p^{-1} = q^{-1} \bmod p$ ,  $a_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod p$  and  $b_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_k^{b_{ik}}) \bmod q$ . Thus, instead of computing  $y_i$  directly with multiplication operations in  $\bmod n$ , a signer first computes  $a_i$  and  $b_i$  with multiplication operations in, respectively,  $\bmod p$  and  $\bmod q$ . Then  $y_i$  is computed from  $a_i$  and  $b_i$ . Since multiplication operations in  $\bmod p$  and  $\bmod q$  are more efficient than in  $\bmod n$ , the signing time is decreased.

This Chinese Remainder Theorem improvement can only be used by a signer because knowledge of the factors of modulus  $n$  is required. A few hundred bytes of additional memory are needed for storing a few large integers (for 512-bit modulus).

3) *Precomputation (precomp)*: A signer can further speed up the signing operation by using more memory. To illustrate the basic idea of this improvement, consider the signing operation with  $k = 4$ . To sign a message, a signer computes  $y_i = r_i \times (s_1^{b_{i1}} \times \dots \times s_4^{b_{i4}}) \bmod n$ , for  $i = 1, \dots, t$ . Since  $s_1, \dots, s_4$  do not change from message to message, and  $b_{i1}, \dots, b_{i4}$  are either one or zero, the signer can precompute and store the product ( $\bmod n$ ) of every nonempty subset of  $\{s_1, \dots, s_4\}$ . Let  $S_{b_{i1} \dots b_{i4}}$  denote the precomputed product  $s_1^{b_{i1}} \times \dots \times s_4^{b_{i4}} \bmod n$ . Then, to sign a message, the signer simply computes  $y_i$  by  $r_i \times S_{b_{i1} \dots b_{i4}} \bmod n$ .

For large  $k$ , it is not practical to precompute the product ( $\bmod n$ ) of every nonempty subset of  $\{s_1, \dots, s_k\}$ . Instead, the signer partitions  $\{s_1, \dots, s_k\}$  into smaller sets and precomputes each of them. If each smaller set contains four  $s_i$ , then it is a 4-bit precomputation. Similarly, if each smaller set contains eight  $s_i$ , then it is an 8-bit precomputation. For 4-bit precomputation with  $k = 128$  and 512-bit modulus, a signer needs to store  $128/4 \times (2^4 - 1) = 480$  products. That is, additional memory of  $480 \times 512$  bits or 31 kB is required. The additional memory required by 8-bit, 12-bit, and 16-bit precomputation are 261 kB, 2.88 MB, and 33.6 MB, respectively.

<sup>12</sup>In practice, for  $k$  up to 128, the verification key components  $\{v_i\}$  are less than  $2^{10}$ , and each component can be stored in 16 bits.

TABLE III  
eFFS SIGNING TIME (MILLISECONDS) WITH 512-BIT MODULUS

	eFFS parameter ( $k, t$ )					
	(32, 1)	(32, 2)	(64, 1)	(32, 4)	(64, 2)	(128, 1)
basic FFS	3.95	7.87	7.21	15.62	14.35	13.72
small v-key	3.95	7.84	7.21	15.63	14.36	13.72
crt + small v-key	3.13	6.20	5.35	12.44	10.63	9.78
4-bit precomp + crt + small v-key	1.95	3.84	2.99	7.61	5.92	5.08
8-bit precomp + crt + small v-key	1.47	2.87	2.02	5.67	3.98	3.14

TABLE IV  
eFFS VERIFICATION TIME (MILLISECONDS) WITH 512-BIT MODULUS

	eFFS parameter ( $k, t$ )					
	(32, 1)	(32, 2)	(64, 1)	(32, 4)	(64, 2)	(128, 1)
basic FFS	3.65	7.07	7.12	14.01	13.63	13.44
small v-key	0.33	0.62	0.43	1.21	0.81	0.65
4-bit precomp + small v-key	0.32	0.60	0.41	1.16	0.76	0.59
8-bit precomp + small v-key	0.32	0.59	0.40	1.14	0.74	0.57

Although a similar precomputation can be used in verification, it is not effective with the small v-key extension. This is because when small primes are used as public key components, their products can be computed very efficiently.

4) *Performance Comparison* We implemented the three speedup extensions using the large integer arithmetic routines from CryptoLib [8]. Tables III and IV show the times for signing and verifying (with 512-bit modulus) 128-bit message digests using different speedup extensions for different values of  $(k, t)$ .<sup>13</sup> The results were obtained on a Pentium II 300 MHz machine running Linux. Note that, for a fixed  $kt$  product, the signing/verification time is smaller when  $t$  is smaller.

In the experiments to be reported in the balance of this paper, we used *8-bit precomp + crt + small v-key* for eFFS signing, and *small v-key only* for eFFS verification.

### C. Adjustable and Incremental Verification

In multicast or group communications, receivers typically have different amounts of resources, and the resources available to a receiver for verification vary over time. It is thus desirable to have an adjustable and incremental signature verification operation. With this extension, a signature can be verified at different security levels. An adjustable verification allows a receiver to verify a message at a lower security level using less resources. An incremental verification allows a receiver to verify a message at a lower security level first, and later increase the security level by using more resources (e.g., if the message is important).

Since the security level of a signature scheme depends on its parameters, e.g., the modulus size, an obvious approach to provide adjustable and incremental verification is to use multiple keys (with different modulus sizes) to generate multiple signatures for different security levels. To verify at a lower security level, the verification key with a shorter modulus size is used to verify the corresponding signature. This approach is simple but very inefficient. In the following, we design

<sup>13</sup>For basic FFS, we specified signing key components  $\{s_i\}$ . Verification key components  $\{v_i\}$  were chosen by CryptoLib.

TABLE V  
eFFS  $t$ -LEVEL SIGNATURE SIGNING TIMES (MILLISECONDS)

	$kt$ product		
	$kt = 32$	$kt = 64$	$kt = 128$
1-level signature	1.47	2.02	3.14
2-level signature		2.87	3.98
4-level signature			5.67

an extension to FFS that provides adjustable and incremental verification efficiently.

Our extension to provide adjustable and incremental verification is to use  $t$  greater than one, and to include  $\{x_i\}$  for  $i = 2, \dots, t$  in signatures. This is called a  $t$ -level signature.<sup>14</sup> This extension is as secure as the original scheme because  $x_i = y_i^2 \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = 2, \dots, t$  can be computed easily from the original signature, which consists of  $\{b_{ij}\}$  and  $\{y_i\}$ , together with the verification key  $\{v_1, \dots, v_k, n\}$  which is publicly known.

To verify a  $t$ -level signature of message  $m$  at security level  $l$  of  $t$  (where  $l \leq t$ ), a verifier does the following: (1) compute  $z_i = y_i^2 \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = 1, \dots, l$  and (2) verify that  $z_2, \dots, z_l$  are equal to  $x_2, \dots, x_l$ , respectively, and the first  $k \times t$  bits of  $h(m, z_1, x_2, \dots, x_t)$  are equal to the  $\{b_{ij}\}$  received.

To increase the verification security level from  $l_1$  to  $l_2$ , a verifier does the following: 1) compute  $z_i = y_i^2 \times (v_1^{b_{i1}} \times \dots \times v_k^{b_{ik}}) \bmod n$  for  $i = l_1 + 1, \dots, l_2$ , and 2) verify that  $z_{l_1+1}, \dots, z_{l_2}$  are equal to  $x_{l_1+1}, \dots, x_{l_2}$ , respectively.

The size of a  $t$ -level signature is  $kt + (2t - 1) \times |n|$  bits. For 512-bit modulus and product  $kt = 128$ , a 1-level signature is 80 bytes and a 2-level signature is 208 bytes.

Table V shows different  $t$ -level signature signing times. For the same  $kt$  product, the signing time increases as the  $t$  value increases. However, the signing time is still smaller than using multiple keys to implement different security levels. For example, the 2-level signature signing time, which is 3.98 ms for  $kt = 128$ , is smaller than the time to sign two (original

<sup>14</sup>Note that the original (1-level) signature does not provide adjustable and incremental verification.

TABLE VI  
eFFS INCREMENTAL VERIFICATION TIMES (MILLISECONDS) FOR  $kt = 128$ .  
(a) 2-LEVEL SIGNATURE. (b) 4-LEVEL SIGNATURE.

To	level 1	level 2
From level 0	0.42	0.81
From level 1		0.40

(a)

To	level 1	level 2	level 3	level 4
From level 0	0.34	0.63	0.93	1.22
From level 1		0.30	0.60	0.89
From level 2			0.30	0.60
From level 3				0.31

(b)

1-level) signatures, one for  $(k, t) = (64, 1)$  and the other for  $(k, t) = (128, 1)$ , which is  $2.02 + 3.14 = 5.16$  ms.

Table VI shows the (incremental) verification times from one level to a higher level for a 2-level signature and a 4-level signature with  $kt = 128$ . In particular, for a 2-level signature, a verifier can first verify a message at level 1 of 2 using 0.42 ms processor time, and later increase to level 2 (of 2) by using 0.40 ms additional processor time.

#### IV. COMPARISON WITH OTHER SIGNATURE SCHEMES

In this section, we compare eFFS(128, 1) to FFS(128, 1) as well as four other signature schemes available from CryptoLib [8], namely: DSA [15], ElGamal [8], RSA [19], and Rabin [17]. We compare their key and signature sizes, and signing and verification times. Then, we compare their signing and verification rates for 1024-byte packets when each is used as the signature scheme in our flow signing and verification procedures presented in Section II. Experiments were performed on a Pentium II 300-MHz machine running Linux. Four different modulus sizes, 384, 512, 768, and 1024 bits, were used in the comparison. (Note that it is difficult to compare the security levels of different signature schemes even if they use the same modulus size.)

##### A. Key and Signature Sizes

Table VII shows the signing/verification key and signature sizes. The signing keys are from 96 to 384 bytes in all schemes except FFS and eFFS whose signing keys are much larger, from 6192 to 16512 bytes. Note that a signing key is private to a signer. We do not expect the relatively large eFFS signing keys to pose a problem for sources/signers of packets.<sup>15</sup>

In RSA and Rabin, verification keys are from 48 to 128 bytes. In DSA, ElGamal, and eFFS, verification keys are slightly larger, from 144 to 404 bytes. Even for receivers with limited resources, we believe that a verification key as large as 400 bytes would not pose a problem. (Note that without the small v-key extension, FFS verification keys are as large as signing keys.)

<sup>15</sup>Such signing keys are, indeed, too large for small devices, such as smartcards, but it is unlikely that these devices would be sources of packet flows or multicasts.

TABLE VII  
SIGNING KEY, VERIFICATION KEY, AND SIGNATURE SIZES (BYTES) OF DIFFERENT SIGNATURE SCHEMES. (a) SIGNING KEY SIZES (BYTES). (b) VERIFICATION KEY SIZES (BYTES). (c) SIGNATURE SIZES (BYTES).

	modulus size (bits)			
	384	512	768	1024
RSA	96	128	192	256
Rabin	96	128	192	256
DSA	136	168	232	296
ElGamal	144	192	288	384
FFS(128,1)	6192	8256	12384	16512
eFFS(128,1)	6192	8256	12384	16512

(a)

RSA	48	64	96	128
Rabin	48	64	96	128
DSA	164	212	308	404
ElGamal	144	192	288	384
FFS(128,1)	6192	8256	12384	16512
eFFS(128,1)	304	320	352	384

(b)

RSA	48	64	96	128
Rabin	48	64	96	128
DSA	40	40	40	40
ElGamal	96	128	192	256
FFS(128,1)	64	80	112	144
eFFS(128,1)	64	80	112	144

(c)

The signature of DSA is the smallest and is 40 bytes for all modulus sizes. For all of the other schemes, the signatures are larger and about the same size, 48 to 256 bytes. In particular, the signature sizes of eFFS and the popular RSA are about the same.

##### B. Signing and Verification Times

Table VIII shows the signing and verification times for a 16-byte message (digest). DSA and ElGamal have been designed to achieve efficient signing (e.g., for use in smartcard applications), and RSA and Rabin have been designed to achieve efficient verification. From Table VIII, note that the signing operations of DSA and ElGamal, with times from 3.9 to 18.9 ms, are much more efficient than those of RSA and Rabin, with times from 6.2 to 95.9 ms. On the other hand, the verification operations of RSA and Rabin, with times from 0.14 to 1.14 ms, are much more efficient than those of DSA and ElGamal, with times from 5.1 to 350.3 ms.

Note that the signing and verification operations of FFS are both inefficient. However, eFFS has a signing operation even more efficient than those of DSA and ElGamal, and a verification operation as efficient as that of RSA. This combination of the most efficient signing and highly efficient verification makes eFFS the best choice for most applications.

##### C. Flow Signing and Verification Rates

Table IX shows the flow signing and verification rates of our flow signing and verification procedures (for 1024-byte pack-

TABLE VIII  
SIGNING AND VERIFYING TIMES (MILLISECONDS) OF DIFFERENT  
SIGNATURE SCHEMES. (a) SIGNING TIME (MILLISECONDS).  
(b) VERIFICATION TIME (MILLISECONDS).

	modulus size (bits)			
	384	512	768	1024
RSA	6.2	12.7	36.2	79.4
Rabin	11.3	19.5	47.5	95.9
DSA	3.9	5.6	10.2	16.3
ElGamal	5.1	6.8	12.3	18.9
FFS(128,1)	8.8	13.7	22.9	38.5
eFFS(128,1)	2.3	3.1	5.2	8.2

(a)

	modulus size (bits)			
	384	512	768	1024
RSA	0.26	0.40	0.70	1.1
Rabin	0.14	0.20	0.38	0.56
DSA	5.1	7.6	14.7	24.2
ElGamal	24.4	51.9	157.5	350.3
FFS(128,1)	8.5	13.4	22.1	37.3
eFFS(128,1)	0.53	0.65	0.82	1.1

(b)

TABLE IX  
FLOW SIGNING AND VERIFICATION RATES (PACKETS/S) FOR 1024-BYTE PACKETS,  
DEGREE TWO TREE CHAINING, AND BLOCK SIZE SIXTEEN. (a) FLOW SIGNING  
RATE (PACKETS/S). (b) FLOW VERIFICATION RATE (PACKETS/S).

	modulus size (bits)			
	384	512	768	1024
RSA	1940	1090	413	193
Rabin	1200	739	321	163
DSA	2760	2140	1320	874
ElGamal	2320	1850	1140	749
FFS(128,1)	1550	1070	624	395
cFFS(128,1)	3920	3140	2160	1610

(a)

	modulus size (bits)			
	384	512	768	1024
RSA	7480	7030	6060	5290
Rabin	7960	7610	7010	6430
DSA	2270	1660	949	609
ElGamal	600	295	99	45
FFS(128,1)	1590	1150	633	419
eFFS(128,1)	6640	6370	5760	5250

(b)

ets, degree-two tree chaining, block size sixteen, and 100% of processor time of a Pentium II 300-MHz machine). Both DSA and ElGamal have low flow verification rates, rendering them inappropriate for receivers with limited resources, such as personal digital assistants and low-end notebook computers. Both RSA and Rabin have low flow signing rates, rendering them inappropriate for real-time generated flows, such as live video/audio applications. By comparison, eFFS provides high flow signing rates suitable for real-time generated flows while its flow verification rates are also very high.

## V. CONCLUSION

We investigated the problem of signing/verifying delay-sensitive packet flows to provide data authenticity, integrity,

and nonrepudiation for Internet applications. We have designed flow signing and verification procedures, based upon a tree-chaining technique, to meet the following requirements: 1) flow signing is efficient and, for real-time generated flows, delay-bounded; 2) flow verification is efficient (for receivers with limited resources); 3) packets in a flow are individually verifiable (for best-effort multicast delivery); 4) packet signatures are small (for a small communication overhead); and 5) verification at a receiver is adjustable to different security levels and can be carried out incrementally (for receivers with limited resources).

We implemented our flow signing and verification procedures and performed experiments to compare different chaining techniques. From experimental results, we recommend the use of degree-two (binary) tree chaining since it requires the smallest packet signature size (i.e., smallest communication overhead) while its signing and verification rates are comparable to the rates of other chaining techniques. Our flow signing and verification procedures are very efficient and achieve one to two orders of magnitude improvement compared to the sign-each approach.

Since signed packets in our procedures are individually verifiable, the procedures can be used to reduce the workload of any machine that sends out a large number of signed packets to one or more destinations. There is no requirement that these packets belong to flows. However, for packets that belong to a flow, the workload of the flow's receiver(s) is also reduced.

To further improve our procedures, we propose several extensions to the Feige-Fiat-Shamir digital signature scheme [3], [4] to speed up both the signing and verification operations, as well as to allow adjustable and incremental verification. The extended scheme, called eFFS, is compared to four other digital signature schemes, RSA [19], Rabin [17], DSA [15], and ElGamal [6], on the same computing platform (Pentium II 300-MHz machine running Linux).

The signing operation of eFFS is by far the most efficient of all the schemes compared. The verification operation of eFFS is as efficient as that of RSA (tie for a close second behind the verification operation of Rabin). In addition to efficient signing and verification, we have extended the eFFS scheme to allow a receiver to efficiently carry out adjustable and incremental verification. Such a capability is useful for large-scale multicast applications with a variety of receivers including some with limited resources.

## APPENDIX

### FLOW VERIFICATION PROCEDURE

procedure flowverify()

for each received packet

if the block signature  $sign(root)$  in the packet signature is new then

/\* this is the first received packet in the block \*/

compute the packet digest;

compute each ancestor of the packet digest

as the message digest of its children;

let  $root'$  be the computed block digest;

if  $(verify(root', sign(root)) = \text{false})$  then

the packet is not verified

```

else
    the packet is verified;
    cache all computed nodes and their children
    as verified
endif
else /* this is not the first received packet in the block */
    compute the packet digest;
    if (packet digest has been cached) then
        if (computed packet digest  $\neq$  its cached value)
            then the packet is not verified
        else
            the packet is verified
        endif
    else
        compute all noncached ancestors of the
        packet digest;
        let node be the highest node computed;
        compute the parent of node;
        if (computed parent  $\neq$  its cached value) then
            the packet is not verified
        else
            the packet is verified;
            cache all computed nodes and their children
            as verified
        endif
    endif
endif
endfor

```

## ACKNOWLEDGMENT

The authors would like to thank the anonymous reviewers for their constructive comments.

## REFERENCES

- [1] T. Ballardie, *Scalable Multicast Key Distribution*, RFC 1949, May 1996.
- [2] S. E. Deering, "Multicast routing in internetworks and extended LANs," in *Proc. ACM SIGCOMM '88*, pp. 55-64.
- [3] U. Feige, A. Fiat, and A. Shamir, "Zero knowledge proofs of identity," in *Proc. 19th Ann. ACM Symp. Theory of Computing*, New York, NY, 1987, pp. 210-217.
- [4] A. Fiat and A. Shamir, "How to prove yourself: Practical solutions to identification and signature problems," in *Advances in Cryptology—CRYPTO '86*, 1987, pp. 186-194.
- [5] S. Floyd, V. Jacobson, C.-G. Liu, S. McCann, and L. Zhang, "A reliable multicast framework for light-weight sessions and application level framing," in *Proc. ACM SIGCOMM '95*, Cambridge, MA, pp. 342-356.
- [6] T. El Gamal, "A public-key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in Cryptology—CRYPTO '84*, 1985, pp. 10-18.
- [7] R. Gennaro and P. Rohatgi, "How to sign digital streams," in *Advances in Cryptology—CRYPTO '97*, 1997, pp. 180-197.
- [8] J. B. Lacy, D. P. Mitchell, and W. M. Schell, "CryptoLib: Cryptography in software," in *Proc. USENIX: 4th UNIX Security Symposium*, Santa Clara, CA, Oct. 1993, pp. 1-17.
- [9] L. Lamport, "Constructing digital signatures from a one-way function," SRI Int., Menlo Park, CA, Tech. Rep. CSL 98, 1979.
- [10] R. C. Merkle, "A digital signature based on a conventional encryption function," in *Advances in Cryptology—CRYPTO '87*, 1987, pp. 369-378.
- [11] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology—CRYPTO '89*, 1989, pp. 218-238.
- [12] S. Micali and A. Shamir, "An improvement on the Fiat-Shamir identification and signature scheme," in *Advances in Cryptology—CRYPTO '88*, 1990, pp. 244-247.
- [13] S. Mitra, "Iolus: A framework for scalable secure multicasting," in *Proc. ACM SIGCOMM '97*, Cannes, France, 1997, pp. 277-288.
- [14] S. Mitra and T. Y. C. Woo, "A flow-based approach to datagram security," in *Proc. ACM SIGCOMM '97*, Cannes, France, 1997, pp. 221-234.
- [15] *National Institute of Standards and Technology*. Digital Signature Standard, NIST FIPS PUB 86, U.S. Dept. Commerce, May 1994.
- [16] C. Partridge, *Using the Flow Label Field in IPv6*, RFC 1809, June 1995.
- [17] M. O. Rabin, "Digitized signatures and public-key functions as intractable as factorization," MIT Lab. Comput. Sci., Cambridge, MA, Tech. Rep. LCS/TR-212, 1979.
- [18] R. L. Rivest, *The MD5 Message Digest Algorithm*, RFC 1321, Apr. 1992.
- [19] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120-126, 1978.
- [20] C. K. Wong, M. Gouda, and S. S. Lam, "Secure group communications using key graphs," in *Proc. ACM SIGCOMM '98*, Vancouver, B.C., 1998, pp. 68-79.
- [21] L. Zhang, S. E. Deering, D. Estrin, S. Shenker, and D. Zappala, "RSVP: A new resource ReSerVation Protocol," *IEEE Network Mag.*, vol. 9, no. 5, pp. 8-18, Oct. 1993.



**Chung Kei Wong** (S'88) received the B.Eng. degree from the University of Hong Kong, Hong Kong, and the M.Phil. degree from the Hong Kong University of Science and Technology. He is currently pursuing the Ph.D. degree in computer sciences and working on multicast and data flow security at the University of Texas at Austin.



**Simon S. Lam** (S'71-M'74-SM'80-F'85) received the B.S.E.E. degree (with Distinction) from Washington State University, Pullman, in 1969, and the M.S. and Ph.D. degrees in engineering from the University of California at Los Angeles (UCLA) in 1970 and 1974, respectively.

From 1971 to 1974, he was a Postgraduate Research Engineer at the ARPA Network Measurement Center, UCLA, where he worked on satellite and radio packet switching networks. From 1974 to 1977, he was a Research Staff Member at the IBM

T. J. Watson Research Center, Yorktown Heights, NY. Since 1977, he has been on the faculty of the University of Texas at Austin, where he is a Professor of Computer Sciences. He holds two anonymously endowed professorships, and served as Department Chair from 1992 to 1994. His research interests in networking include protocol and switch design, performance analysis, distributed multimedia, quality of service guarantees, and security.

Dr. Lam has served on the editorial boards of IEEE/ACM TRANSACTIONS ON NETWORKING, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, IEEE TRANSACTIONS ON COMMUNICATIONS, PROCEEDINGS OF THE IEEE, and *Performance Evaluation*. He was Editor-in-Chief of IEEE/ACM TRANSACTIONS ON NETWORKING from 1995 to 1999. He organized and was Program Chair of the inaugural ACM SIGCOMM Symposium held at the University of Texas at Austin in 1983. He is a founding Steering Committee member of the IEEE International Conference on Network Protocols. He received the 1975 Leonard G. Abraham Prize Paper Award from the IEEE Communications Society for his paper on packet switching in a multiaccess broadcast channel. He is a Fellow of the Association for Computing Machinery (ACM).